Three-Loop Temporal Interpolation for Error Concealment of MDC

Mengyao Ma†, Oscar C. Au‡, S.-H. Gary Chan†, Liwei Guo‡ and Zhiqin Liang‡
† Dept. of Computer Science, ‡ Dept. of Electrical and Electronic Engineering
Hong Kong University of Science and Technology
{myma, eeau, gchan, eeglw, zhiqin}@ust.hk

***Abstract*— *Multiple Description Coding* (MDC) can be used as an *Error Resilience* (ER) technique for video coding. In case of transmission errors, *Error Concealment* can be combined with MDC to reconstruct the lost frame, such that the propagated error to the following frames is reduced. In this paper, we propose a new temporal error concealment method named *Three-loop Temporal Interpolation* (TLTI). TLTI can be well combined with temporal sub-sampling ER methods, such as MDC and *Alternative Motion-Compensated Prediction*. In the simulation, we compare the performance of TLTI with *Unidirectional Motion Compensated Temporal Interpolation* (UMCTI). Both visual and quantitive results show that TLTI can achieve a better video quality than UMCTI.**

I. INTRODUCTION

Error Resilience (ER) and *Error Concealment* (EC) techniques are very important for video transmission today, due to the use of predictive coding and *Variable Length Coding* (VLC) in video compression [1]. The conventional INTER mode approach is illustrated in Figure 1(a), where each P-frame is predicted from its immediate previous frame. Although the compression efficiency of this approach is high, it is vulnerable to errors in the transmission channel. If one frame is lost or corrupted (for example: P_4) during the transmission, the error in the reconstructed frame at the decoder will propagate to the remaining frames until the next I-frame (I_{11}) is received.

Several ER methods have been developed for video communication, such as *Forward Error Correction* (FEC) [2], *Layered Coding* [3], and *Multiple Description Coding* (MDC) [4]. Different from the traditional *Single Description Coding* (SDC), MDC divides the video stream into equally important *streams* (*descriptions*), which are sent to the destination through different channels. One simple implementation is the odd/even sub-sampling approach: an even (odd) frame is predicted from the previous even (odd) frame, as illustrated in Figure 1(b). Since the reference frames are farther in time, the prediction of such approach is not as good as the conventional codec and the compression efficiency is lower. On the other hand, since each stream is encoded and transmitted separately, the corruption of one stream will not affect the other. As a result, the decoder can simply display the correct video stream ($P_5P_7P_9\ldots$) at half of the original frame rate, or reconstruct the corrupted frame by some appropriate EC methods, e.g. *Temporal Interpolation*. The objective of using temporal interpolation is that it can be well combined with temporal MDC methods. Recall that in Figure 1(c), when

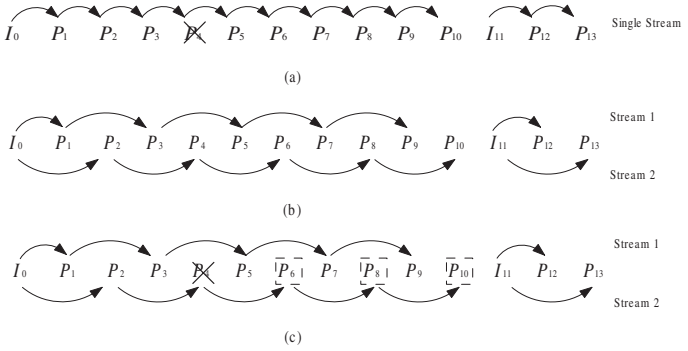


Fig. 1. Illustration of different approach for video coding (the arrow means that the previous frame is used as the reference of the latter). (a) Conventional video coding; (b) Odd/even sub-sampling MDC; (c) Error occurs in (b).

frame P_4 is corrupted during the transmission, its surrounding frames (P_3 and P_5) would be correct if stream 1 is error-free. So we can utilize P_3 and P_5 to interpolate P_4 with good quality.

Temporal interpolation was originally used to generate one or more frames between two received frames so as to improve the effective frame rate, and make the object motions in the video smoother. A *Motion Compensated Temporal Interpolation* (MCTI) method is proposed in [5], which uses block-based motion estimation to track motions of the objects between adjacent received frames. Improved methods are proposed in [6] and [7], to remove the blocking artifact. The common feature of these methods is that both forward and backward motion estimation are performed to find the motion vector, which lead to high computational requirement. In [8], *Unidirectional Motion Compensated Temporal Interpolation* (UMCTI) is used, which performs only forward motion estimation, and thus saves half of the computation time. Motivated by UMCTI, we propose an error concealment algorithm called *Three-loop Temporal Interpolation* (TLTI). TLTI utilizes the preserved motion vector in the correct stream for the concealment, and can be well combined with temporal sub-sampling ER methods, such as MDC and *Alternative Motion-Compensated Prediction* (AMCP) [9][10][11].

The rest of this paper is organized as follows. In Section 2, we describe the proposed approach TLTI. The performances of TLTI and UMCTI are compared in Section 3, in terms of both PSNR and visual quality. Section 4 is conclusion.

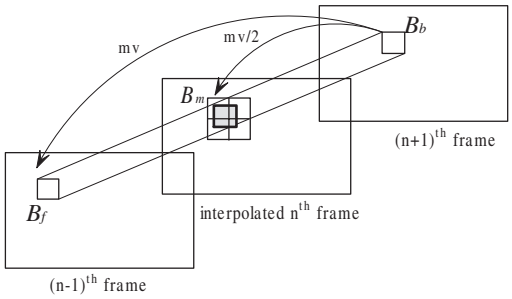


Fig. 2. Illustration of temporal interpolation.

II. Three-loop Temporal Interpolation

When the odd/even sub-sampling is used in temporal MDC, an even frame is predicted from the previous even frame, and an odd frame is predicted from the previous odd frame. Then these two streams are sent to the decoder through different channels. Suppose the failure probability of each channel is independent. Then if the n^{th} frame is lost during the transmission, its neighboring frames may be correct, which can be used to reconstruct it by temporal interpolation. In this section, we will introduce the proposed method *Three-loop Temporal Interpolation* (TLTI), which is motivated by UMCTI in [8]. The advantage of introducing UMCTI to temporal MDC is that the exhaustive motion estimation can be prevented, since the motion vectors from blocks of the $(n+1)^{th}$ frame to the corresponding blocks in the $(n-1)^{th}$ frame are known. In other words, the motion vector from P_5 to P_3 is conserved in stream 1, as in the example of Figure 1(c).

Instead of dividing the lost frame into 16×16 blocks as in UMCTI, we use 4×4 block size, not only because smaller block size can reduce the blocking artifact, but also it can adapt to the multiple block sizes of H.264 [12]. Each 4×4 block has two motion vectors: one is the determined motion vector for the final interpolation mv_d, and the other one is the candidate motion vector mv_c; both of them are initialized to be *Undefined Number*, e.g. ∞. The proposed method needs three loops to fill the pixel values of the n^{th} frame:

1) Determine mv_d of possible blocks: As illustrated in Figure 2, each 4×4 block (B_b) in the $(n+1)^{th}$ frame has a motion vector mv pointing to the $(n-1)^{th}$ frame. If the motion is linear translation, the corresponding block in the n^{th} frame should be B_m, shaded area indicated by $\frac{1}{2}mv$. As B_m may not align the gird, it can overlap more than one blocks. We divide the blocks in the n^{th} frame into two sets: set $\mathcal{O}$ contains the blocks which overlap with the region, indicated by $\frac{1}{2}mv$ of some block in the $(n+1)^{th}$ frame; set $\mathcal{N}$ contains the remaining blocks. For any block $B_i^n \in \mathcal{O}$, we want to find its motion vector $mv_{d_i}^n$ to the $(n-1)^{th}$ frame for the interpolation.[1] One way is to use the motion vector of the block, which has the largest overlapped region with B_i^n:

$$mv_{d_i}^n = \frac{1}{2} mv_m^{n+1}, \text{where } size_i(m) = \max_{B_j^{n+1} \in \mathcal{P}^i} size_i(j). \quad (1)$$

[1]For B_i^n, n indicates a block in the n^{th} frame and i is its index in set $\mathcal{O}$.

Here $\mathcal{P}^i$ is the set of blocks in the $(n+1)^{th}$ frame whose $\frac{1}{2}mv$ pointing to an area overlapped with B_i^n; the motion vector of B_j^{n+1} is mv_j^{n+1}, and $size_i(j)$ is the overlapped region size between B_i^n and the area indicated by $\frac{1}{2}mv_j^{n+1}$. However, experimental result shows that it is not stable to only use the overlapped region size to determine the motion vector. One reason is that the maximum region size is sometimes too small to be credible; another reason is the motion vector preserved in the $(n+1)^{th}$ frame is not reliable, due to the unknown motion estimation method in the encoder side. So we change the definition of $\mathcal{P}^i$ to be $\mathcal{P}^{i*} =$

$$\{B_j^{n+1} | B_j^{n+1} \in \mathcal{P}^i \ \& \ \|mv_j^{n+1}\| \leq \text{MV}_t \ \& \ size_i(j) \geq \text{SIZE}_t\},$$

where SIZE_t and MV_t are two thresholds. $mv_{d_i}^n$ is still determined by (1), with $\mathcal{P}^i$ replaced by $\mathcal{P}^{i*}$.

After the modification of $\mathcal{P}^i$, the mv_ds of some blocks in set $\mathcal{O}$ may not be defined in the first loop, due to an empty $\mathcal{P}^{i*}$. For these blocks, we can save their candidate motion vectors. In stead of using the overlapped region size to decide mv_c, we use the smoothness of the motion vectors of neighboring blocks (top, down, left, right) as the selection criterion. Suppose $B_i^n \in \mathcal{O}$; its candidate motion vector is determined by

$$mv_{c_i}^n = \frac{1}{2} mv_l^{n+1}, \text{where } MD_i(l) = \min_{B_j^{n+1} \in \mathcal{P}^i} MD_i(j). \quad (2)$$

Here $MD_i(j)$ is the minimum *Euclidean Distance* between $\frac{1}{2}mv_j^{n+1}$ and the mv_ds of the four neighbors of B_i^n. If one neighbor does not exist, or its mv_d is not defined, the *Euclidean Difference* is defined to be ∞.

In the implementation of this algorithm, one way to determine the motion vectors of the blocks in $\mathcal{O}$ is maintaining two lists for each block: one for mv_d and the other one for mv_c. Then the values of mv_d are determined first using (1), followed by the determination of mv_c using (2). The disadvantage of this approach is that large memory is needed to save all the possible motion vectors. So we propose to use another method requiring less memory: each block only needs one more buffer (sz) to save the overlapped region size. The algorithm works as follows: visit all the blocks in the $(n+1)^{th}$ frame, from top to bottom and from left to right. For any block B_b, we can find its corresponding blocks in the n^{th} frame using $\frac{1}{2}mv$, as in Figure 2. For each of these blocks (at most 4), we can update its mv_d using the criteria in (1). Since only one mv_d is saved, buffer sz is needed to save the largest overlapped region size. If mv_d can not be determined, i.e. $\mathcal{P}^{i*}$ is empty, criteria in (2) can be used to update mv_c. Although this implementation needs less memory, the value of mv_c of some blocks may be incorrect after the first loop. Take $mv_{c_i}^n$ as an example, its value is related to the mv_ds of the four neighbors of B_i^n, which may be changed after the updating of $mv_{c_i}^n$. As a result, $mv_{c_i}^n$ is different from the one got from (2). We will solve this problem in the third loop.

2) Fill blocks with defined mv_d: After the first loop, some blocks in set $\mathcal{O}$ have defined mv_d, we can fill the pixel values

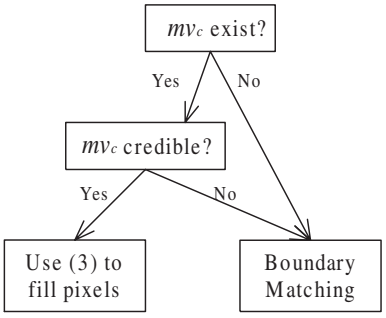


Fig. 3. Illustration of the third loop of TLTI.

of such blocks using:

$$p^n(i,j) = \frac{1}{2}[p^{n-1}(i+dx, j+dy) + p^{n+1}(i-dx, j-dy)], \quad (3)$$

where $p^n(i,j)$ is the pixel value of the n^{th} frame at position (i,j), and (dx,dy) is the vector representation of mv_d. These filled blocks can help the *Boundary Matching* in the next loop [13].

3) Fill other blocks: For the remaining unfilled blocks after the previous loop, there are two ways for the concealment, depending on whether mv_c is available. Figure 3 is the illustration. As noted previously, the mv_cs of some blocks may not be correct after the first loop. So for each block with mv_c, we first test whether its mv_c is *Credible*, i.e. the smoothness of the motion vectors between the current block and its neighbors still holds:

$$\min_{i \in \{u,d,l,r\}} \|mv_c - mv_{di}\| \leq \Delta_t, \quad (4)$$

where u, d, l, r are the indexes of the four neighbors, and mv_{di} represents their determined motion vectors used in filling pixels; Δ_t is a threshold. In case one neighbor does not exist, or it has not been filled, the *Euclidean Difference* is defined to be ∞. After the smoothness testing, if mv_c is *Credible*, (3) can be used to fill the pixel values, with $(dx,dy) = mv_d = mv_c$; otherwise, *Boundary Matching* can be used.

Boundary Matching (BM) was first proposed in [13], which estimates the lost motion vector using minimum boundary variance as the criteria. We use both the forward and the backward frames as the references, and the average boundary variance of the four neighboring blocks (top, down, left, right) are calculated, if available. The motion search is preformed within a search range (16×16), using the median motion vector of the neighboring blocks (up, left, up-left) as the initial value. After the motion search, the average of the target blocks in the two references are used to fill the pixel values. Note that BM does not perform well when the block boundary is a horizontal/vertical edge. In the implementation, we first check whether the adjacent blocks have such edges. For example, if the upper block has a horizontal line at the bottom, it is not used in the calculation of boundary variance. *Sobel* operator is used to check the horizontal/vertical line in the area of the reference frame, indicated by the motion vector of the checked block.

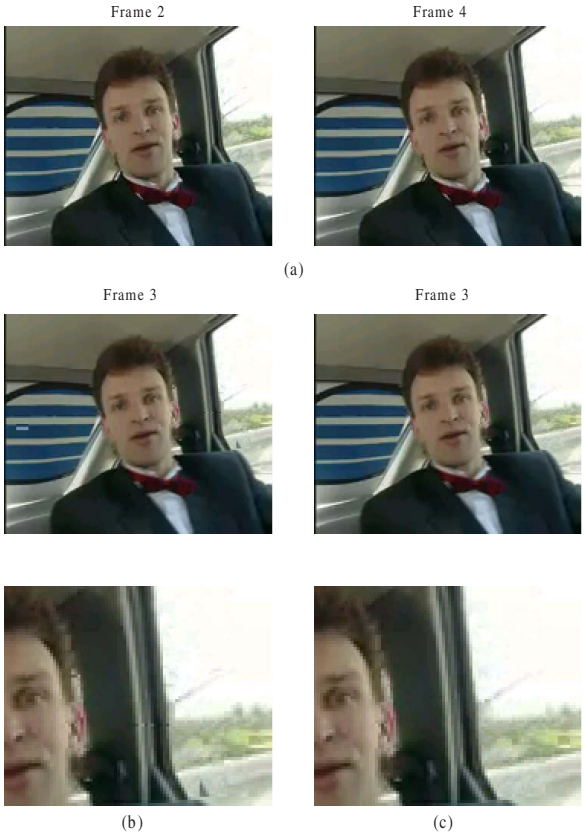
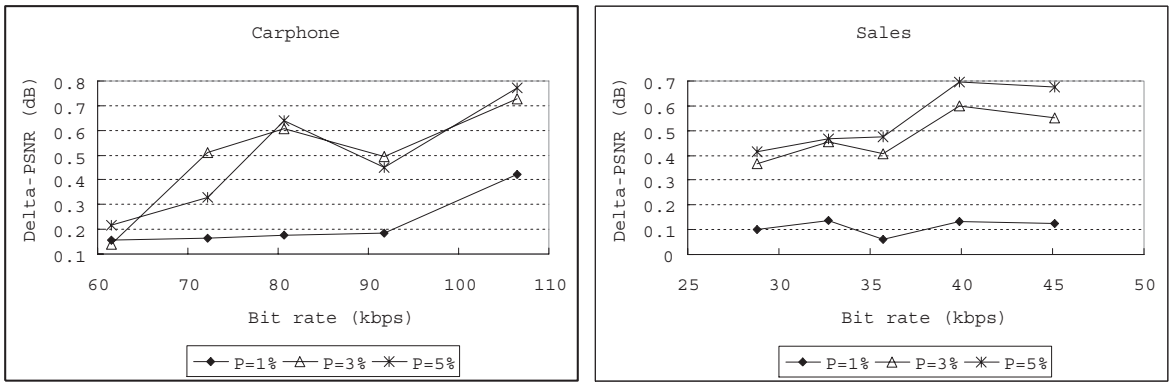


Fig. 4. The visual result of applying UMCTI and TLTI on *Carphone*, for one frame loss (frame 3). (a) The original encoded frames without loss; (b) Concealed frame using UMCTI (PSNR=29.79dB); (c) Concealed frame using TLTI (PSNR=30.33dB). For (b) and (c), the bottom pictures are the enlarged right parts of the corresponding upper pictures.

III. Simulation Results

In the simulation, we compare the performance of TLTI with UMCTI, by both visual and quantitive results. We use the JVT reference software version 8.2 (baseline profile) for the simulations [14]. The first 300 frames of video sequences *Carphone* and *Sales* (QCIF) are encoded at 15fps, and only the first frame is I frame. At the encoder side, ref_idx_l0 is specified for each P frame to simulate the odd/even sub-sampling MDC. For the I frame, we just send it twice to the decoder side, since the main focus of the simulation is to compare the performance of temporal interpolation, instead of the compression efficiency of MDC. To further improve the coding of I frame, method in [15] can be employed. During the concealment, constant thresholds are used for TLTI: $\text{SIZE}_t = 8$, $\text{MV}_t = 3\sqrt{2}$ and $\Delta_t = 3\sqrt{2}$. For UMCTI, we also use the preserved motion vector in the correct stream for the interpolation, thus reducing the computation time.

Figure 4 illustrates the visual quality after applying UMCTI and TLTI on *Carphone*, for one frame loss (frame 3). Fixed QP is used for the encoding, 27 for I frame and 29 for P frame. The first row lists the correctly received frames, and the second row is the reconstructed frames by UMCTI and TLTI, respectively. The bottom two pictures are the enlarged



(a) (b)

Fig. 5. The average delta-PSNR between TLTI and UMCTI for different packet loss rate ($P = 1\%$, $P = 3\%$ or $P = 5\%$).

right parts of the corresponding upper pictures. From these figures we can see that the concealed frame using TLTI looks much better than that using UMCTI, especially around object boundaries. Less blocking artifacts are introduced.

We also compare the performance of UMCTI and TLTI, under random packet loss condition. Suppose the failure probability of each channel is independent and identically distributed with probability P; $P = 1\%$, 3% and 5% are used. One packet contains the information of one frame, and the loss of one packet will lead to the loss of one entire frame. Five different bit rates are selected for the compression of each sequence. For each combination of loss rate (P) and bit rate, we transmit the video sequence 40 times. At the decoder side, UMCTI or TLTI is used to reconstruct the lost frames, and the average PSNR is computed by comparing with the original encoded one. Note that these two algorithms work for the condition of one frame loss, i.e., the surrounding two frames are received from the other channel and reconstructed with/without error. For continuous losses at the decoder side, as far as we know, there is no good error concealment method in the literature. In such cases, copying previous frame (*Freeze*) can be used to reconstruct the video. I addition, if the lost frame is the last one of a GOP, *Freeze* is applied. The delta-PSNR between TLTI and UMCTI is obtained for the 40 transmissions, and its average value is plotted in Figure 5. We can see that in all the testing cases, TLTI can obtain a higher average PSNR than UMCTI, especially when the loss rate is higher.

IV. Conclusion

In this paper, we propose a new error concealment method for the odd/even sub-sampling MDC, named *Three-loop Temporal Interpolation* (TLTI). The good feature of TLTI is that it can be well combined with temporal sub-sampling ER methods, such as MDC and AMCP [9][10][11]. Simulation results show that TLTI can reconstruct the lost frame with a better quality than UMCTI. Note that in the current work, constant thresholds are used for TLTI. In the future, we will investigate a set of variable thresholds, which may adaptively change the values according to the statistics of each block, and thus improve the interpolated video quality.

Acknowledgment

This work has been supported in part by the Innovation and Technology Commission (projects no. ITS/122/03 and GHP/033/05) and the Research Grant Council (DAG04/05.EG34) of the Hong Kong Special Administrative Region, China.

References

[1] Y. Wang and Q. F. Zhu, "Error control and concealment for video communication: a review," in *Proc. IEEE*, May 1998, pp. 974 – 997.

[2] Y. Mei, W. Lynch, and L. N. Tho, "Joint forward error correction and error concealment for compressed video," in *Proc. IEEE ITCC*, Apr. 2002, pp. 410 – 415.

[3] C.-M. Fu, W.-L. Hwang, and C.-L. Huang, "Efficient post-compression error-resilient 3D-scalable video transmission for packet erasure channels," in *Proc. IEEE ICASSP*, Mar. 2005, pp. 305 – 308.

[4] Y. Wang, A. Reibman, and S. Lin, "Multiple description coding for video delivery," in *Proc. IEEE*, Jan. 2005, pp. 57 – 70.

[5] C.-K. Wong and O. Au, "Fast motion compensated temporal interpolation for video," in *Proc. SPIE VCIP*, May 1995, pp. 1108 – 1118.

[6] C.-K. Wong, O. Au, and C.-W. Tang, "Motion compensated temporal interpolation with overlapping," in *Proc. IEEE ISCAS*, May 1996, pp. 608 – 611.

[7] T. Chen, "Adaptive temporal interpolation using bidirectional motion estimation and compensation," in *Proc. IEEE ICIP*, Sept. 2002, pp. 313 – 316.

[8] C.-W. Tang and O. Au, "Unidirectional motion compensated temporal interpolation," in *Proc. IEEE ISCAS*, June 1997, pp. 1444 – 1447.

[9] J. Apostolopoulos, "Reliable video communication over lossy packet networks using multiple state encoding and path diversity," in *Proc. SPIE VCIP*, Jan. 2001, pp. 392 – 409.

[10] S. Wenger, "Video redundancy coding in h.263+," in *Proc. Audio- Visual Services over Packet Networks*, Sept. 1997.

[11] M. Ma, O. C. Au, and S.-H. G. Chan, "A new motion compensation approach for error resilient video coding," in *Proc. IEEE ICIP*, Sept. 2005, pp. I–773–776.

[12] G. Sullivan and T. Wiegand, "Video compression - from concepts to the H.264/AVC standard," *Proc. IEEE*, vol. 93, pp. 18 – 31, Jan. 2005.

[13] W. Lam, A. Reibman, and B. Liu, "Recovery of lost or erroneously received motion vectors," in *Proc. IEEE ICASSP*, Apr. 1993, pp. 27–30.

[14] Jvt reference software, version 8.2. [Online]. Available: http://iphome.hhi.de/suehring/tml/download/

[15] Y. Wang, M. Orchard, and A. Reibman, "Multiple description image coding for noisy channels by pairing transform coefficients," in *IEEE MMSP*, June 1997, pp. 419 – 424.